Search Engine Architecture

7. Classification
Agenda

• Introduction to machine learning
• Gradient descent
• Logistic regression
• Stochastic gradient descent
• Scaling optimization
Machine Learning: High-Level

- Algorithms that can learn from data
- Supervised
  - Infer a function from labeled training data
- Unsupervised
  - No labels in training data
  - Find interesting patterns or structure

Supervised Machine Learning

• The generic problem of function induction given sample instances of input and output
  • Classification: output draws from finite discrete labels
  • Regression: output is a continuous value
• Focus here on supervised classification
  • Suffices to illustrate large-scale machine learning

This is not meant to be an exhaustive treatment of machine learning!

Applications

- Spam detection
- Content (e.g. movie) classification
- POS tagging
- Friendship recommendation
- Document ranking
- Many, many more!

scikit-learn algorithm cheat-sheet

classification
- SVC
- Ensemble Classifiers
- KNeighbors Classifier
- SGD Classifier
- Naive Bayes
- Text Data
- Linear SVC
- kernel approximation

<100K samples

regression
- SGD Regressor
- Lasso
- ElasticNet
- SVR(kernel="rbf")
- ensemble Regressors
- RidgeRegression
- SVR(kernel="linear")

>50 samples

predicting a category

predicting a quantity

<100K samples

just looking

predicting structure

<10K samples

dimensionality reduction
- Isomap
- Spectral Embedding
- LLE

<10K samples

<10K samples

<10K samples

number of categories known

tough luck

few features should be important

start

got more data

SIMPLE

di you have labeled data

<100K samples

Supervised Binary Classification

- Restrict output label to be *binary*
  - Yes/No
  - 1/0
- Binary classifiers form a primitive building block for multi-class problems
  - One vs. rest classifier ensembles
  - Classifier cascades

Limits of Supervised Classification?

• Why is this a big data problem?
  • Isn’t gathering labels a serious bottleneck?
• Solution: user behavior logs
  • Learning to rank
  • Computational advertising
  • Link recommendation
• The virtuous cycle of data-driven products

The Task

Given $D = \{(x_i, y_i)\}_{i=1}^n$

$x_i = [x_1, x_2, x_3, \ldots, x_d]$

$y \in \{0, 1\}$

Induce $f : X \to Y$

Such that loss is minimized

$$
\arg \min_{\theta} \frac{1}{n} \sum_{i=0}^{n} \ell(f(x_i; \theta), y_i)
$$

Typically, consider functions of a parametric form:

Key insight: machine learning as an optimization problem!
(closed form solutions generally not possible)

Gradient Descent: Preliminaries

- Rewrite:
  \[ \arg \min_{\theta} \frac{1}{n} \sum_{i=0}^{n} \ell(f(x_i; \theta), y_i) \quad \rightarrow \quad \arg \min_{\theta} L(\theta) \]

- Compute gradient:
  - “Points” to fastest increasing “direction”
  \[ \nabla L(\theta) = \left[ \frac{\partial L(\theta)}{\partial w_0}, \frac{\partial L(\theta)}{\partial w_1}, \ldots, \frac{\partial L(\theta)}{\partial w_d} \right] \]

- So, at any point:
  \[ b = a - \gamma \nabla L(a) \]
  \[ L(a) \geq L(b) \]

Gradient Descent: Iterative Update

• Start at an arbitrary point, iteratively update:
  \[ \theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \nabla L(\theta^{(t)}) \]

• We have:
  \[ L(\theta^{(0)}) \geq L(\theta^{(1)}) \geq L(\theta^{(2)}) \ldots \]

• Lots of details:
  • Figuring out the step size
  • Getting stuck in local minima
  • Convergence rate

Gradient Descent

Repeat until convergence:

\[ \theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(x_i; \theta^{(t)}), y_i) \]
Intuition behind the math...

\[ \ell(x) \]

\[ \frac{d}{dx} \ell \rightarrow \nabla \ell \]

\[ \theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(x_i; \theta^{(t)}), y_i) \]

New weights  Old weights  Update based on gradient

Gradient Descent

\[ \theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(x_i; \theta^{(t)}), y_i) \]
Lots More Details...

- Gradient descent is a “first order” optimization technique
  - Often, slow convergence
  - Conjugate techniques accelerate convergence
- Newton and quasi-Newton methods:
  - Intuition: Taylor expansion
    \[
    f(x + \Delta x) = f(x) + f'(x)\Delta x + \frac{1}{2} f''(x)\Delta x^2
    \]
  - Requires the Hessian (square matrix of second order partial derivatives): impractical to fully compute

Logistic Regression
Logistic Regression: Preliminaries

• Given \( D = \{(x_i, y_i)\}_i^n \)
  \[ x_i = [x_1, x_2, x_3, \ldots, x_d] \]
  \[ y \in \{0, 1\} \]

• Let’s define:
  \[ f(x; w) : \mathbb{R}^d \rightarrow \{0, 1\} \]
  \[ f(x; w) = \begin{cases} 1 & \text{if } w \cdot x \geq t \\ 0 & \text{if } w \cdot x < t \end{cases} \]

• Interpretation:
  \[
  \ln \left[ \frac{\Pr(y = 1|x)}{\Pr(y = 0|x)} \right] = w \cdot x \\
  \ln \left[ \frac{\Pr(y = 1|x)}{1 - \Pr(y = 1|x)} \right] = w \cdot x
  \]

Why log odds?

• Dot product alone would be unbounded (not a probability)
  • Solve using odds
• Empirically, we see diminishing returns
  • Solve using log

\[
\ln \left[ \frac{\Pr(y = 1|x)}{1 - \Pr(y = 1|x)} \right] = w \cdot x
\]

Relation to the Logistic Function

- After some algebra:

\[
\Pr(y = 1 | x) = \frac{e^{w \cdot x}}{1 + e^{w \cdot x}}
\]

\[
\Pr(y = 0 | x) = \frac{1}{1 + e^{w \cdot x}}
\]

- The logistic function:

\[
f(z) = \frac{e^z}{e^z + 1}
\]

Training an LR Classifier

- Maximize the conditional likelihood:
  \[ \arg \max_w \prod_{i=1}^n \Pr(y_i|x_i, w) \]

- Define the objective in terms of conditional log likelihood:
  \[ L(w) = \sum_{i=1}^n \ln \Pr(y_i|x_i, w) \]

- We know \( y \in \{0, 1\} \) so:
  \[ \Pr(y|x, w) = \Pr(y = 1|x, w)^y [1 - \Pr(y = 0|x, w)]^{(1-y)} \]

- Substituting:
  \[ L(w) = \sum_{i=1}^n \left( y_i \ln \Pr(y_i = 1|x_i, w) + (1 - y_i) \ln \Pr(y_i = 0|x_i, w) \right) \]

LR Classifier Update Rule

• Take the derivative:

\[
L(w) = \sum_{i=1}^{n} \left( y_i \ln \Pr(y_i = 1|x_i, w) + (1 - y_i) \ln \Pr(y_i = 0|x_i, w) \right)
\]

\[
\frac{\partial}{\partial w} L(w) = \sum_{i=0}^{n} x_i \left( y_i - \Pr(y_i = 1|x_i, w) \right)
\]

• General form for update rule:

\[
w^{(t+1)} = w^{(t)} + \gamma^{(t)} \nabla_w L(w^{(t)})
\]

\[
\nabla L(w) = \left[ \frac{\partial L(w)}{\partial w_0}, \frac{\partial L(w)}{\partial w_1}, \ldots, \frac{\partial L(w)}{\partial w_d} \right]
\]

• Final update rule:

\[
w_{i}^{(t+1)} = n_{i}^{(t)} + \gamma^{(t)} \sum_{j=0}^{n} x_{j, i} \left( y_j - \Pr(y_j = 1|x_j, w^{(t)}) \right)
\]

Lots more details...

- Overfitting
  - Model tuned too tightly to training data
- Regularization
  - E.g. add sum of square weights to loss function (L2)
- Different loss functions
  - E.g. hinge, epsilon-insensitive
- Evaluation
  - Accuracy, precision, recall, F1, ...
    - Sound familiar?
- Cross-validation

MapReduce Implementation

\[ \theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(x_i; \theta^{(t)}), y_i) \]

iterate until convergence

compute partial gradient

mappers

single reducer

update model

Shortcomings

- MapReduce is bad at iterative algorithms
  - Hadoop has high job startup costs
  - Awkward to retain state across iterations
- High sensitivity to skew
  - Iteration speed bounded by slowest task
- Potentially poor cluster utilization
  - Must shuffle all data to a single reducer
- Some possible tradeoffs
  - Number of iterations vs. complexity of computation per iteration
  - E.g., L-BFGS: faster convergence, but more to compute

Scaling Optimization
Stochastic Gradient Descent
Batch vs. Online

Gradient Descent

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \frac{1}{n} \sum_{i=0}^{n} \nabla \ell(f(x_i; \theta^{(t)}), y_i)$$

“batch” learning: update model after considering all training instances

Stochastic Gradient Descent (SGD)

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \nabla \ell(f(x; \theta^{(t)}), y)$$

“online” learning: update model after considering each (randomly-selected) training instance

“mini-batch” learning: compromise between batch and fully online

In practice... just as good!

Ensembles
Ensemble Learning

- Learn multiple models, combine results from different models to make prediction
- Why does it work?
  - If errors uncorrelated, multiple classifiers being wrong is less likely
  - Reduces the variance component of error
- A variety of different techniques:
  - Majority voting
  - Simple weighted voting:
    \[ y = \arg \max_{y \in Y} \sum_{k=1}^{n} \alpha_k p_k(y|x) \]
  - Model averaging

Practical Notes

• Common implementation:
  • Train classifiers on different input partitions of the data
  • Embarrassingly parallel!
• Contrast with bagging
  • Samples chosen randomly with replacement
• Contrast with boosting
  • Shift weight toward samples with errors

MapReduce Implementation

\[
\theta^{(t+1)} \leftarrow \theta^{(t)} - \gamma^{(t)} \nabla \ell(f(x; \theta^{(t)}), y)
\]

Sentiment Analysis Case Study

Lin and Kolcz, SIGMOD 2012

- Binary polarity classification: {positive, negative} sentiment
  - Independently interesting task
  - Illustrates end-to-end flow
  - Use the “emoticon trick” to gather data

- Data
  - Test: 500k positive/500k negative tweets from 9/1/2011
  - Training: {1m, 10m, 100m} instances from before (50/50 split)

- Features: Sliding window byte-4grams

- Models:
  - Logistic regression with SGD (L2 regularization)
  - Ensembles of various sizes (simple weighted voting)

Ensembles with 10m instances better than 100m single classifier!  

"for free"

Rundown

- Ridge regression
  - MSE loss with L2 regularization
- LASSO
  - MSE loss with L1 regularization
- Random forest
  - Bagging, subsampled features
- Gradient boosting
  - Train new models on residuals
- Support vector machine
  - Hinge loss, L2 regularization

Scaling Stochastic Gradient Descent

(From Jeff Dean’s RecSys2014 Keynote)
Neural Networks

Input:

Output:
2012 Supervised Vision Model: “AlexNet”

- Softmax to predict object class
- Fully-connected layers (and trained w/ DropOut)
  - ~60M parameters

Basic architecture developed by Krizhevsky, Sutskever & Hinton
Won 2012 ImageNet challenge with 16.4% top-5 error rate
Vision models, 2014 edition: GoogLeNet

Module with 6 separate convolutional layers

24 layers deep!

No fully-connected layer, so only ~6M parameters (but ~2B floating point operations per example)

Developed by team of Google Researchers:
Won 2014 ImageNet challenge with 6.66% top-5 error rate
Model Parallelism: Partition model across machines
Data Parallelism: Asynchronous Distributed Stochastic Gradient Descent

Parameter Server: \[ p' = p + \Delta p \]

Model Workers

Data Shards
Staleness

Parameter Server

Model Workers

Data Shards

Too old! Ignore!

Very slow

\( \Delta \rho \) \hspace{1cm} \rho
Questions?